Int J Digit Libr (2000) 3: 208–220 / Digital Object Identifier (DOI) 10.1007/s007990000036

Original Articles

On Z39.50 wrapping and description logics*

Yannis Velegrakis[1,**], **Vassilis Christophides**[2], **Panos Constantopoulos**[2,3]

[1] Department of Computer Science, University of Toronto, 10 King's College Rd, Toronto, Ontario, Canada M5S-3G4;
E-mail: velgias@cs.toronto.edu
[2] Institute of Computer Science, FORTH, Vassilika Vouton, P.O. Box 1385, GR 711 10, Heraklion, Greece;
E-mail: {christop, panos}@ics.forth.gr
[3] Department of Computer Science, University of Crete, GR 71409, Heraklion, Greece

Published online: 22 September 2000 –

Abstract. Z39.50 is a client/server protocol widely used in digital libraries and museums for searching and retrieving information spread over a number of heterogeneous sources. To overcome semantic and schematic discrepancies among the various data sources the protocol relies on a world view of information as a flat list of fields, called *Access Points* (AP). One of the major issues for building Z39.50 wrappers is to map this unstructured list of APs to the underlying source data structure and semantics. For highly structured sources (e.g., database management systems, knowledge base systems) this mapping is quite complex and considerably affects the quality of the retrieved data. Unfortunately, existing Z39.50 wrappers have been developed from scratch and they do not provide high-level mapping languages with verifiable properties. In this paper, we propose a description logic (DL) based toolkit for the declarative specification of Z39.50 wrappers. We claim that the conceptualization of AP mappings enables a formal validation of the query translation quality (e.g., ill-defined mappings, inappropriate APs, etc.) and allows one to tackle a number of Z39.50 pending issues (e.g., metadata retrieval, query failures due to unsupported APs, etc.). Furthermore, our DL-based approach allows the development of Z39.50 wrappers enriched with a number of added-value services such as conceptual structuring of flat Z39.50 vocabularies and intelligent Z39.50 query assists. These services are quite useful for profile developers, Z39.50 wrappers administrators, and end-users.

Key words: Information retrieval – Z39.50 wrapping – Description logics – Query rewriting

* This work was partially supported by the European project AQUARELLE (Telematics Application Programme IE-2005) and the CIMI Interoperability Testbed project.
** Work done while the author was at the ICS-FORTH.

1 Introduction

With the advances in digital processing and communication technologies an increasing number of organizations and individuals are using the Internet for publishing, broadcasting, and exchanging information all over the world. The ability to share and manipulate information from multiple sources is a fundamental requirement for large-scale applications, e.g., digital libraries and museums. A widely used protocol for searching and retrieving information in a distributed environment is Z39.50 [1]. To achieve interoperability [50], Z39.50 (Version 3) relies on: (i) standard messages, formats, and procedures governing the communication of clients and servers (*system interoperability*); (ii) a world view of information as a flat vocabulary of fields, called *Access Points* (AP), that abstracts representational details of source data (*semantic* and *schematic* interoperability); and (iii) basic textual search primitives to express Boolean queries in the form of field-value pairs (*query interoperability*).

In order to evaluate Z39.50 queries, sources should wrap their actual data organization, format, and query capabilities according to the Z39.50 specifications established for a specific application, community, etc. These specifications are described in the various *profiles* (i.e., metadata) proposed by national or international bodies (e.g., Library of Congress[1], CIMI[2], etc.). It should be stressed that the quality of the established mappings between the source and the Z39.50 view of information is fundamental in order to ensure the quality of the retrieved data (i.e., accuracy, consistency, completeness, etc.). Unfortunately, existing Z39.50 wrappers are developed using some programming language and they do

[1] lcweb.loc.gov
[2] www.cimi.org

not provide abstract mapping languages with verifiable properties [10, 51, 52]. In this paper, we advocate a Description Logics[3] framework [8] for the declarative specification of Z39.50 wrappers using high-level concept languages. We claim that modeling the required mappings as first-class citizens, instead of hard coding them in the wrappers: (i) allows the formal validation of the translation quality (e.g., ill-defined mappings, inappropriate APs); and (ii) opens unexpected opportunities to tackle a number of Z39.50 pending issues (e.g., metadata retrieval, query failures due to not mapped APs, multiple answer sets handling, etc.).

Building a wrapper for an information source according to a Z39.50 profile (e.g., for digital libraries [37, 38], museums [23, 53], etc.) implies the translation of: (i) the Z39.50 *Access Points* to the underlying source data structure and semantics; (ii) the Z39.50 Boolean filters to the source query language; and (iii) the returned source data from their original format to a predefined Z39.50 record syntax (e.g., GRS-1, XML). For loosely structured sources (e.g., information retrieval systems) wrapping is relatively simple. It essentially requires the definition of some renaming mappings from the APs to the source attributes or tags (e.g., the AP *AU* to the field *author*, etc.). However, for highly structured sources (e.g., database management systems, knowledge base systems) this translation is considerably more complex. This is mainly due to the significant mismatch that exists between the Z39.50 flat view of information and the underlying source data model (e.g., relation or class based). In this context, what is really needed is to define for each AP a view on the source data.

To address this issue we introduce an intermediate level between the Z39.50 and the source world, based on advanced knowledge representation and reasoning support, specifically *Description Logics* (DL). DL provide declarative languages to represent and reason about interrelated sets of objects using modeling primitives such as concepts, roles, and individuals. Starting from a set of primitive concepts and roles representing source data organization, we capture the semantics of the AP mappings as derived concepts formed by primitive ones and standard DL concept constructors [4]. Since a DL can serve both as a knowledge representation language and as a query language [7, 12, 49], derived concepts essentially act as views [14] against which Z39.50 queries can be evaluated with the source data. Our contribution is twofold: (i) we propose a toolkit for the declarative specification of Z39.50 wrappers using standard DL reasoning mechanisms [25]; and (ii) we enrich the so developed Z39.50 wrappers with a number of added-value services such as conceptual structuring of flat Z39.50 vocabularies and intelligent Z39.50 query processing. As we will see, these services are quite useful for profile developers, Z39.50 wrappers administrators, and end-users.

[3] dl.kr.org

The rest of the paper is organized as follows. In Sect. 2 we give an example of a cultural information source and describe the encountered problems to wrap it according to a digital museum Z39.50 profile. In Sect. 3 we briefly recall the core DL we use for the declarative specification and validation of Z39.50 AP mappings. Section 4 presents the Z39.50 query processing in our DL setting and Sect. 5 elaborates on the offered added-value wrapping services. The architecture of the Z39.50 wrapper toolkit is presented in Sect. 6. Related work in Z39.50 wrapping and DL-based mapping languages is presented in Sect. 7. Finally, we conclude and discuss future work in Sect. 8.

2 An example of a cultural information source

In this section we describe the contents and structure of a cultural information source that will be used as running example in the rest of the paper. We focus on the mismatch of the information conceptualization in our test database and a Z39.50 profile for Digital Museums [23, 53], as well as on the problems we have encountered in order to develop a Z39.50 wrapper in the context of the AQUARELLE and CIMIzit projects [44, 45].

2.1 The CLIO system

Our testbed relies on the CLIO cultural documentation system, developed at the Institute of Computer Science, Foundation for Research and Technology-Hellas (ICS-FORTH) in close cooperation with the Benaki Museum, Athens and the Historical Museum of Crete, Heraklion. CLIO supports the management of an evolving body of knowledge about ensembles of cultural goods and addresses the needs of museum curators and researchers. The functional kernel of CLIO is the Semantic Index System (SIS) developed by ICS-FORTH [24]. SIS is a persistent storage system based on the object-oriented semantic network data model TELOS [46].

Figure 1 illustrates some modeling primitives of our SIS-based source namely *multiple classification* as well as *multi-valued* and *optional attributes*. A museum object is represented as an instance of the class "MuseumObject". It may have (*optional attributes*) an owner (class "Owner") and be constructed with the use of one or more (*multi-valued attributes*) materials (class "Material"), processes (class "Process"), and techniques (class "Technique"). Each museum object is associated to a series of events (class "Event") characterized by their kind, date, and involved actor. For instance, the saber of Androutsos (a hero of the 1821 Hellenic Revolution) is made of shaped silver (*multiple instantiation*) and it was constructed by Filimon in 1815. Although not illustrated in our example, SIS-TELOS also supports *simple* and *multiple inheritance*, *unbounded classification*, and treats *attributes as first-class citizens* classified on their own.

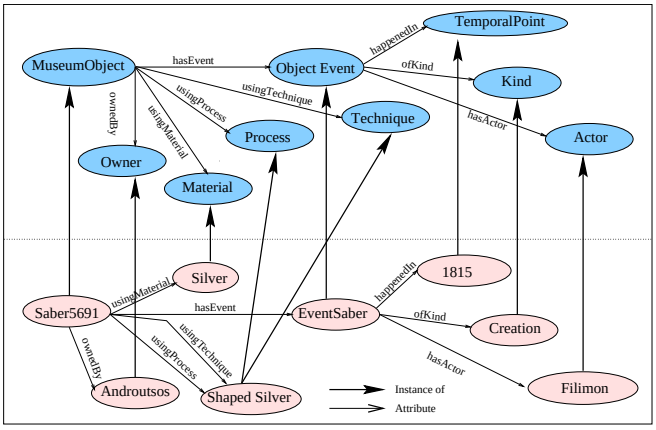

Fig. 1. An example of a cultural information source

2.2 Z39.50 wrapping for digital museums

Z39.50 [1] is a session oriented and stateful application protocol, based on standard client-server architecture. To overcome semantic and schematic discrepancies among the various data sources, Z39.50 relies on a common information model shared by all clients and servers. It consists of a flat list of fields, called *Access Points* (AP) (or more precisely *Use Attributes*), on which queries are expressed. For instance, in the CIMI [23] and in the AQUARELLE [53] profile, the supplied APs correspond to general information categories like *People* (specific persons or cultural groups), *Date*s of many sorts (including dates of creation, acquisition, exhibition), *Place*s (e.g., place of creation or provenance, places associated with an event, etc.), *Subject* (exact description of depicted material), *Style* (including movement and period), *Method* (including process and techniques), *Material*, etc. (see [34] for further details).

This vocabulary of fields is employed by a Z39.50 client in order to search for records in the underlying sources and next, to retrieve some or all of them. Z39.50 queries are formulated using Boolean connectors (*and*, *or*, *and-not*), search terms (i.e., use attribute-value pairs), and qualifiers specifying lexicographical comparisons (e.g., greater than), truncations (e.g., right, left), etc. For instance, the following query searches for all the museum objects related with Androutsos and created after 1887:

Q1: `PersonalName`="Androutsos" **and**
(`DateOfCreation`=1887 `Relation`="GreaterThan")

To answer **Q1** a Z39.50 server should translate it into the native query language of our testbed source (i.e., SIS). This translation requires a *wrapping module* able, in a first step, to map the involved APs to the corresponding classes of our cultural scenario. According to Fig. 1, the person *Androutsos* might be the creator (i.e., the actor involved in a creation event), or the owner of the object. Then, the wrapper should translate a query on the AP *PersonalName* into source queries on the *Actor* and *Owner* classes. Similarly, a query on the AP *DateOfCreation* should be translated into queries on the *TemporalPoint* class and the associated *Object_Event* and *Kind* classes. Finally, the retrieved museum objects information from our source, should be formatted/converted by the wrappers according to a common agreed record syntax (e.g., GRS-1, XML) and structure (e.g., including the elements *ObjectId*, *Title*, etc.). These records will be delivered by a Z39.50 server to the requesting clients.

The APs translation is relatively simple if the source is an information retrieval system (IRS). We believe that the underlying Z39.50 information model is more suitable to query loosely structured text bases than highly structured data sources (e.g., DBMS, KBS). Indeed, due to the significant mismatch between the Z39.50 and the source models, most of the existing structure and semantic richness of the sources is not taken into account during querying, while wrapping becomes considerably more complicated. It becomes clear that an AP may be translated to a source query on one or more classes or relations using one or more attribute selections, joins, etc. In other words, for each AP we essentially need to define a view over the source data. Nothing guarantees that the semantics of these views correspond to the intended meaning of the APs in the Z39.50 profile: it may be included, partially overlapped, etc. This is typically the kind of information that is missing from existing Z39.50 servers in order to ensure the quality of the retrieved data (i.e., consistency, accuracy, completeness, etc.). As we will see in the next sections the definition of AP views (i.e., mappings) using formal languages allows reasoning about the quality of the Z39.50 query translation. Two Z39.50 wrapping issues are worth further elaboration and they will be addressed by our DL approach.

2.2.1 Unsupported access points

Since the AP meaning is defined in a Z39.50 profile without prior knowledge of the source contents, it may be only implicitly represented in the source or it may not correspond at all to any source information. For example, our cultural source contains objects from the gun collection kept in the *Benaki Museum* and although not explicitly stated, this information could be used to answer queries on the AP *Location*. On the other hand, the AP *Protection Status*, dealing with buildings and monuments, is not at all applicable. According to the protocol both APs are considered as *unsupported* by our source, i.e., the source cannot provide any information about the related AP concepts. According to the protocol queries containing unsupported APs will fail and the Z39.50 server should return a diagnostic message to the requesting client. For large scale applications where queries are generated by a Z39.50 client without a knowledge of servers' metadata (i.e., mappings) it is very likely to exist at least one unsupported AP per source. This will result in embarrassing

query failures and users risk obtaining no answer from the sources. A commonly used approach to cope with this problem is to omit the unsupported APs from the broadcasted query and try to answer only the supported part. Obviously with this approach users are not aware if the returned answers are obtained from the execution of the full query or from a part of it. Moreover, in this solution the semantics given to unsupported APs changes with respect to the kind of the issued query, leading Z39.50 wrappers to behave in an unpredictable manner. For instance, omitting an unsupported AP from a conjunction implies that the AP is interpreted to any source object while in the case of negations the AP is interpreted as the empty set. It should be stressed that the same problems arise when the search operators defined in a Z39.50 profile (e.g., truncations) are not supported by the underlying source.

2.2.2 Fixed collections of retrieved objects

The information returned in response to a client request is always associated with a specific data collection in the source (e.g., a persistence root). In the rest of the paper we will call this root a *central concept*. No matter what the queried APs are, the answer always corresponds to central concept instances (e.g., museum objects). This implies that all the queried fields are supposed to be connected in a source with the Z39.50 central concept. However, this is not the case with structured sources (relational or object-oriented) where multiple collections are supported and data relationships are not always explicitly stated in the schema (using external keys or object paths). Furthermore, even when such paths are explicitly stated, Z39.50 profiles usually support APs for expressing full-text queries that require navigation over sets of paths. For instance, we may use the AP *Any*, to query on term "Androutsos", without specifying what exactly the related APs are: "Androutsos" may correspond in our cultural source to a person owning an object, a person creating an object, a geographical location, etc. Unless the native query language of the source supports *generalized path expressions* [21, 22], these kinds of mappings cannot easily be expressed in structured sources. It is up to the Z39.50 server administrator to decide query evaluation under these circumstances in a more or less ad hoc way.

3 Specifying Z39.50 wrappers using DL

Description Logics (DL), also known as terminological logics, are a family of logics which have been intensively studied for more than a decade in the field of conceptual modeling. DLs provide quite expressive declarative languages for the representation and reasoning about classes of objects and their relationships, encompassing other well-known formalisms such as entity-relationship or class inheritance models [19]. Recently, DLs have received considerable attention in the context of information integration systems [2, 5, 17, 28, 30, 42] since it was proved to provide powerful formalisms to model and reason over a large number of data integration views [43]. We follow the same approach to define the required AP mappings as views over source data. In the sequel, we briefly recall the core DL we use to cope with the various Z39.50 wrapping issues presented previously and provide Z39.50 wrappers with formally verifiable mapping specifications.

3.1 The core description logic

The main DL modeling primitives are concepts, roles, and individuals. A concept describes a class of **individuals** in the domain of interest and it is defined by the conditions that must be satisfied by the individuals to be members of the class. An individual is a description of a real-world entity. A role describes a relationship between two individuals. The two basic components of a DL system are the *terminological box* (**TBox**) and the *assertional box* (**ABox**). The former contains concept descriptions while the latter contains assertions regarding the individuals. There exist two types of concepts: *primitive* and *derived*. The definition of a primitive concept specifies only the necessary conditions for an individual to be a member of it. On the other hand, a derived concept states the necessary and sufficient conditions for an individual to be a member of it. This implies that an individual has to be explicitly defined a member of a primitive concept, while **individuals** of derived concepts can be inferred by the DL system.

The interpretation of a DL knowledge base Σ is $\mathcal{I} = (\mathcal{I}(\Delta), \mathcal{I}(\cdot))$ where $\mathcal{I}(\Delta)$ denotes a non-empty set of individuals (the domain) and $\mathcal{I}(\cdot)$ an interpretation function, mapping every concept to a subset of $\mathcal{I}(\Delta)$, every role to a subset of $\mathcal{I}(\Delta) \times \mathcal{I}(\Delta)$, and every individual to an element of $\mathcal{I}(\Delta)$ such that $\mathcal{I}(a) \neq \mathcal{I}(b)$ for different individuals a, b (*Unique Name Assumption*). Intuitively, the interpretation of a concept C (denoted as $\mathcal{I}(C)$) is the set of individuals that have either been inferred or declared explicitly to be instances of C. A concept C_1 is said to be *subsumed* by another concept C_2 (denoted as $C_1 \dot{\leq} C_2$) if and only if $\mathcal{I}(C_1) \subseteq \mathcal{I}(C_2)$ for all interpretations. Based on this subsumption relation, a set of concepts can form a taxonomy having a *bottom* ($\bot$), and *top* ($\top$) concept. The $\bot$ concept is the concept for which $\mathcal{I}(\bot) \equiv \emptyset$, and the $\top$ the concept for which $\mathcal{I}(\top) \equiv \Delta$.

A **TBox** is defined by a finite set of axioms having one of the forms: $A \dot{\leq} D$ (i.e., $\mathcal{I}(A) \subseteq \mathcal{I}(D)$), ${}_C|R|_D$, and a finite set of concept definitions of the form $K \doteq E$, where A, C, D are primitive concepts, K is a derived concept, R is a role (note the domain and range restrictions of roles), and E is a concept obtained using other concepts and the *constructors* shown in Table 1. Similarly to the roles, we can also have attributes on concepts. Disjointness of primi-

Table 1. Concept- and role-forming operators

Name	Syntax	Semantics
Concept name	A	$\mathcal{I}(A)$
Top	$\top$	Δ
Bottom	$\bot$	$\emptyset$
Union	$A \sqcup C$	$\{d_1 \mid d_1 \in \mathcal{I}(A) \cup \mathcal{I}(C)\}$
Intersect	$A \sqcap C$	$\{d_1 \mid d_1 \in \mathcal{I}(A) \cap \mathcal{I}(C)\}$
Existential quantification	$\exists R.C$	$\{d_1 \mid \exists d_2 : (d_1, d_2) \in \mathcal{I}(R) \wedge d_2 \in \mathcal{I}(C)\}$
Universal quantification	$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in \mathcal{I}(R) \rightarrow d_2 \in \mathcal{I}(C)\}$
Negation	$\neg A$	$\{d_1 \mid d_1 \notin \mathcal{I}(A)\}$
OneOf	$\{i,\ j, ...\}$	$\{\mathcal{I}(i),\ \mathcal{I}(j), ...\}$
Role name	R	$\mathcal{I}(R)$
Role restriction	${}_A\lvert R\rvert_B$	$\{(d_1, d_2) \mid (d_1, d_2) \in (\mathcal{I}(R) \cap (\mathcal{I}(A) \times \mathcal{I}(B)))$
Role reversion	R^{-1}	$\{(d_1, d_2) \mid (d_2, d_1) \in \mathcal{I}(R)$

tive concepts in the **TBox** is given by axioms of the form: $A \| C$ (i.e., $\mathcal{I}(A) \cap \mathcal{I}(C) \equiv \emptyset$). Finally, *cycles* in the concept definitions are not allowed.[4] Hence, an acyclic set of concept definitions can be *unfolded* by iteratively substituting every concept and role name with its definition.

The **ABox** is defined from a finite set of declarations having one of the forms: $C(a)$ and $R(a, b)$. The first one (*unary predicates*) declares that individual a belongs to the interpretation of the concept C and the second one (*binary predicates*) declares that there exists a role R from a to b (respectively belonging to the interpretations of concepts C and D in the definition of R). The main reasoning services [25] offered by a DL system Σ are the following:

- **Concept satisfiability** ($\Sigma \not\models C \equiv \bot$) checking if a concept has no empty interpretation
- **Subsumption checking** ($\Sigma \models C_1 \dot{\leq} C_2$) checking if a concept C_2 subsumes C_1
- **Instance checking** ($\Sigma \models C(a)$) checking if an individual a belongs to the interpretation of concept C.

The above setting corresponds to an almost standard DL framework ($\mathcal{ALEUCOIF}$) [25], actually supported by several DL systems[5] e.g., CRACK,[6] CICLOP,[7] KRIS,[8] etc.

[4] A cycle is defined as a chain of concepts $C_0, C_1, ..., C_n$ such that $C_n = C_0$, and for $0 < k < n-1$ it holds that concept C_{k+1} appears in the definition of concept C_k [47].

[5] The only subtle issue here is the introduction of restricted and inverse roles as in [18, 31].

[6] www.cs.man.ac.uk/ franconi/crack

[7] www-ensais.u-strasbg.fr/LIIA/ciclop/ciclop.htm

[8] www.dfki.uni-sb.de/~tacos/kris.html

3.2 DL concept languages for Z39.50 AP mappings

In a very natural way, source structure and semantics can be represented as primitive concepts and roles [9, 12, 35], and the AP mappings as derived concepts defined on top.

The part of the **TBox** that contains the primitive concepts is usually called the *schema* part while the one that contains the derived concepts is called the *view* part [14]. Figure 2 illustrates the primitive concepts (**TBox**-*schema part*) representing our cultural source schema given in Fig. 1 while the derived concepts (**TBox**-*view part*) correspond to the established mappings for the APs of the CIMI-AQUARELLE profile [23, 53]. The objects of our cultural source are represented by the individuals (**ABox**) of the DL system. Note that this is only a logical view of information from the Z39.50 wrappers (see Sect. 6) and there is no need to actually load source objects into the DL system (virtual **Abox**).

Recall that in the core DL framework previously presented, *cycles* are not allowed in the definition of the derived concepts. However, this restriction does not exclude the existence of cycles in the source schema that can be easily represented by primitive role definitions (e.g., define a role *Related* with domain and range *MuseumObject*). Moreover, the acyclic definition of derived concepts do not limit the expressiveness of our DL setting in order to map APs to source schemas since we do not need to introduce recursive definitions of derived concepts (i.e., views).

In the following examples we illustrate how the proposed DL concept language (see Table 1) can capture the various kinds of translations involved in Z39.50 wrapping for structured sources (see Sect. 2). In addition, as we will see in the next section, this language can be also used to capture the semantics of core Z39.50 Boolean queries.

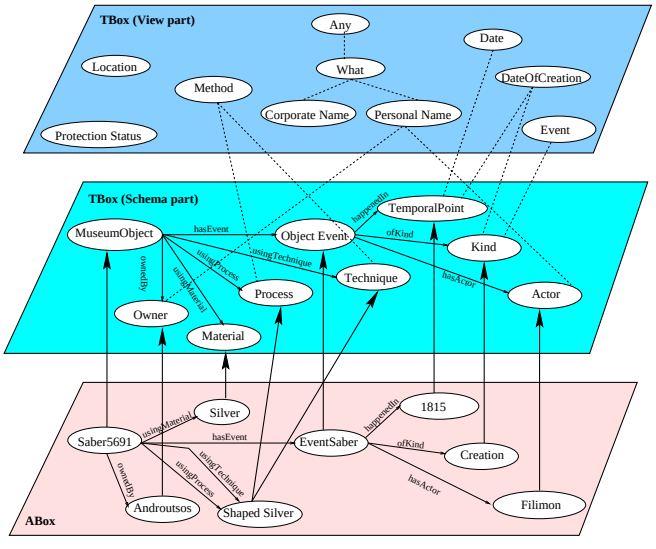

Fig. 2. Modeling an information source and Z39.50 APs mappings in DL

Example 1: Perhaps the simplest case to map an AP is when its semantics corresponds exactly to one concept of the source schema. For instance, the AP *Date* is translated as follows:

$$Date \doteq TemporalPoint$$

Example 2: In most practical cases, APs should be mapped by combining more than one source concepts using the DL *Union* and *Intersect* concept-forming operators. For instance, information about persons in our cultural source is represented by the concepts *Actor* and *Owner*, and the AP *PersonalName* is mapped as follows:

$$PersonalName \doteq Actor \sqcup Owner$$

Similarly, the mapping of the AP *Method* is defined as:

$$Method \doteq Process \sqcap Technique$$

Furthermore, mappings of abstract APs like *Who* describing any personal or corporate name that can be found in our source, are defined by using other AP derived concepts such as:

$$Who \doteq PersonalName \sqcup CorporateName$$
$$Any \doteq Who \sqcup What \sqcup When \sqcup Where$$

Finally, APs like *Any*, for full-text queries are easily mapped by considering the definitions of abstract APs like *Who*, *What*, *When*, and *Where* (the 4W APs).

Example 3: More complicated situations arise when the AP mapping requires a traversal over the roles associated with aggregated source concepts. For example, to map the AP *DateOfCreation* we need to define the following derived concept using the DL *Inverse Role* operator:

$$DateOfCreation \doteq \exists(happenedIn)^{-1}.(\exists ofKind.\{\text{“Creation”}\})$$

The above expression has three parts: (i) the bracket expression corresponds to a concept having as interpretation only the individual Creation, i.e., subsumed by *Kind*; (ii) the parenthesis expression represents the related creation *Event*s; and (iii) the whole expression captures the *Date*s associated with these events. Note that the restriction of a role *to* and *from* values obviates the need to verify that the returned individuals actually belong to the interpretation of *Date*.

Example 4: For APs corresponding to information not explicitly stated in a source, the DL *OneOf* concept-forming operator is used to translate them. For instance, although not given in our example source, it is known that all objects belong to the Benaki Museum (Athens) gun collection, and hence the APs *CorporateName*, *Location*, and *Collection*, are mapped as follows:

$$CorporateName \doteq \{\text{“Benaki Museum”}\}$$
$$Location \doteq \{\text{“Benaki Museum Athens”}\}$$
$$Collection \doteq \{\text{“Benaki Gun Collection”}\}$$

This implies that *CorporateName*, *Location*, and *Collection* are concepts whose interpretation contains only one individual, respectively “Benaki Museum”, “Benaki Museum Athens”, and “Gun Collection”.

Example 5: In the case where there is no information in the source corresponding to a specific AP, the related derived concept is defined to be equivalent either to the *Bottom*, i.e., the concept with an empty interpretation or the *Top*, i.e., the concept whose interpretation contains all the individuals. The decision depends on the expected *precision* and *recall*: the former favors *precision* while the latter *recall*. More precisely, according to the semantics of APs in a Z39.50 profile, we consider that the *Top* for AP mappings is the AP concept *Any* previously defined (see Example 2). For instance, the AP *ProtectionStatus* which is used for preserved buildings and cannot be mapped to our cultural source of museum objects, is translated as follows:

$$ProtectionStatus \doteq \bot \text{ (or } ProtectionStatus \doteq Any)$$

In both cases, wrappers are able to smoothly incorporate unsupported APs into the query processing by avoiding ad hoc omissions of unsupported APs. For instance, the following query is equivalent to the first conjunct only if *ProtectionStatus* is mapped to $\bot$ (see Sect. 5.2):

`PersonalName` = “Androutsos” `and-not`
`ProtectionStatus` = “Preserved”

3.3 Formal validation of Z39.50 wrapping quality

Having defined the mappings of the Z39.50 APs as derived concepts (i.e., views) on top of a source schema, standard DL reasoning services like *Concept Satisfiability* can be used to infer if some or all of the APs mappings are ill-defined. Consider, for instance, that the concept *Material* of our culture source is disjoint with the concepts *Technique* and *Process* (see Fig. 2). Then, the following mapping of the AP *Method* (see Example 2) is inconsistent:

$$Method \doteq Material \sqcap Process \sqcap Technique$$

Indeed, due to the disjointedness, the derived concept *Method* describes a necessarily empty set. In our DL framework we can formally check whether $\Sigma \not\models Method \equiv \bot$, i.e., *Method* has a contradictory description (intentional semantics). More generally, we can verify the consistency of all the established mappings (i.e., that are well defined and not mapped to the bottom) by simply checking whether the **TBox** has at least one model: $\Sigma \not\models$.

To conclude this section we should note that modeling the AP mappings as DL-derived concepts allows one to develop Z39.50 wrappers with formally verifiable properties. More precisely: (i) APs whose meaning is not at all or only implicitly represented in the source can be effectively mapped and smoothly incorporated into the query processing; and (ii) consistency of the established APs mapping can be easily checked without accessing the

source data (virtual **Abox**). These added value services are quite useful for both Z39.50 wrappers' administrators and end-users.

4 Z39.50 query processing using DL

Since DL can serve both as a knowledge representation language and as a query language [7, 12, 49], Z39.50 queries can also be modeled as derived concepts. More precisely, a query can be seen as a description of the necessary and sufficient conditions that have to be satisfied by the individuals corresponding to the objects in the query answer. Conversely, primitive (i.e., source) or derived concepts (i.e., AP mappings) can be used for data querying by considering their interpretation. In the sequel, we show how the Z39.50 Boolean filters can be: (i) translated by the wrappers using the same DL concept language employed to map the Z39.50 APs; and (ii) rewritten by taking into account the defined AP views and the fixed central concept of the objects actually returned by a source (see Sect. 2).

4.1 Translating core Z39.50 queries

As we have seen in Sect. 2, Z39.50 queries are essentially composed of search terms with APs and qualifiers for comparisons, truncations, etc., eventually combined using Boolean connectors. Consider, for instance, the following simple query (i.e., no qualifiers):

Q2: `PersonalName` = "Androutsos"

Recall that *PersonalName* is an AP, mapped as derived concept (C_{AP}) to the *Actor* and *Owner* concepts, and "Androutsos" a value considered as individual (a). **Q2** can be translated into a basic query to the DL knowledge base Σ using the *Instance Checking* reasoning service ($C_{AP}(a)$):

$$\Sigma \models PersonalName(\text{“}Androutsos\text{”})$$

If the individual "*Androutsos*" belongs to the interpretation of the concept *PersonalName* (after unfolding the derived concept to its constituents *Actor* and *Owner* primitive concepts), the knowledge base returns a positive answer and the answer set contains only the individual "*Androutsos*". Otherwise the answer set will be empty. Generally speaking, core Z39.50 queries can be translated into elementary DL queries that correspond to new derived concepts (**Tbox**-*query part*). These query concepts will be evaluated with source individuals (virtual **Abox**) as follows:

Definition 1. *Given a DL knowledge base Σ, an AP-derived concept C_{AP} and a core Z39.50 query q of the form $AP = a$, the answer set of q is given by the interpretation of the concept C_q: $\mathcal{I}(C_q) = \{a \in \mathcal{O}_\Sigma | \; \Sigma \models C_{AP}(a)\}$, where $\mathcal{O}_\Sigma$ denotes the individuals of the **Abox** of Σ.*

Note that query answering relies here on a *closed-world assumption* (CWA) form [32]. In the style of [26] we make the realistic assumption about complete knowledge of the DL assertional part (*closed-domain assumption*) and thus consider in the interpretation of concepts only their known individuals. This assumption is justified by the fact that in our Z39.50 wrapping context, query answering takes place at the source side which provides complete descriptions about its actual objects. We will come back to Z39.50 query answering in the next section.

Now let us see how we can express Z39.50 queries using relation or truncation qualifiers, such as:

Q3: `PersonalName`="Andr" `Truncation`="Right"

These search operators are not directly expressed in a standard DL framework, but they can be captured as external functions. The DL operator **TEST-C** allows one to call various *test functions* outside of a DL system. This operator is essentially an escape method from the limits of the DL expressiveness allowing one to manipulate individuals using external functions written in some programming language (see, e.g., CLASSIC[9] [11]). A *test function f* gets an individual as argument and returns TRUE or FALSE if it satisfies the conditions specified in the body of the function. The interpretation of the expression TEST-C(f) is then all the individuals which, given as argument, the TRUE value is returned by f. Only monotonic functions are considered in this respect. **Q3** can then be translated into the following elementary DL query:

$$\Sigma \models (PersonalName \sqcap \text{TEST-C}(rtrunc_{\text{“}Andr\text{”}}))$$

where $rtrunc_{\text{“}Andr\text{”}}$ is a test function supported by our example source, which performs right truncation on string "Andr".

Finally, the concept-forming operators $\sqcap$, $\sqcup$, and $\neg$ (see Table 1) can be straightforwardly used to capture the Z39.50 Boolean connectors *and*, *or*, and *and-not*.

It should be stressed that when the search operators defined in a Z39.50 profile are not supported by the underlying source, we are confronted with the same problems as in the case of unsupported APs (see Sect. 2.2). To cope with these problems we map unsupported Z39.50 search operators either to the *true* or *false test functions*. The former favors *recall*, since it returns all the individuals of the queried AP concept, while the latter favors *precision*, since it returns the empty set. In both cases, wrappers are able to smoothly incorporate unsupported search operators into the query processing.

4.2 Rewriting Z39.50 queries

Unfortunately, the previous translation into DL is not sufficient to express the exact semantics of Z39.50 queries as defined in a profile. As we have seen in Sect. 2, the result of a Z39.50 query is the set of relevant individuals

[9] www.research.att.com/sw/tools/classic

belonging to a central concept of interest (e.g., the root of museum objects), rather than the set of individuals belonging to the involved AP concepts.

To cope with this problem we need to rewrite the obtained elementary DL queries (or more precisely their unfolded form) in order to take into account the *concept path expressions* (P_{AP}) connecting, through roles, the individuals of the central concept (C_C) with those of the AP-derived concepts involved in a query. C_C is also defined in the **Tbox** as a derived concept (e.g., $C_C \doteq$ *MuseumObject*). For instance, for the AP concept *DateofCreation* used in **Q1** we consider the following path (see Fig. 2):

$P_{CreationDate} \doteq \exists hasEvent.(\exists happenedIn.TemporalPoint)$

Since *DateofCreation* is only a simple case and AP-derived concepts are usually defined by more than one primitive concept (e.g., *PersonalName*), what is really needed is to declare, for each of its constituents concepts (e.g., *Actor*, *Owner*), the corresponding paths to the central concept, e.g., (see Fig. 2):

$P_{PersonalName1} \doteq \exists hasEvent.(\exists hasActor.Actor)$
$P_{PersonalName2} \doteq \exists ownedBy.Owner$

Note that composite AP concepts which are defined in terms of others (e.g., the 4W APs), do not require the definition of paths for their constituent concepts. They are simply inferred during query rewriting. More formally:

Definition 2. *A path expression P_{AP} is a sequence of elements $p = e_1 e_2 \dots e_{n-1} e_n$ such that for $i \in [1, n-1]$: $e_i \in \{\exists\} \cup \{\forall\} \cup \mathcal{R}$., where $\mathcal{R}$ is the set of the primitive role names (suffixed by ".") and $e_n \in \mathcal{C}$ is the set of primitive concepts.*

These paths are used during the Z39.50 query rewriting phase to capture the exact answer set (C_{Answer}) with the individuals of the central concept. It should be stressed that our choice to introduce the concept path expressions during this phase is justified by the fact that the two alternative representations of AP mappings present some serious drawbacks. On one hand, if we model the AP mappings as derived concepts extended with the corresponding paths, we will obtain incorrect Z39.50 queries. Indeed, the used search terms (e.g., `PersonalName` = "Androutsos") will be evaluated against the individuals of the central concept (e.g., *MuseumObject*) instead of the individuals of the involved concepts (e.g., *Owner* and *Actor*). On the other hand, if we model the AP mappings as derived roles connecting through path expressions the involved schema concepts with the central one, we will increase the complexity of the underlying DL [18, 31] (for role instance checking, subsumption, etc.). For these reasons, we consider the following rewriting steps:

1. Core Z39.50 queries are initially translated into elementary DL query concepts as described in the previous section. For instance, the preliminary translation of **Q1** presented in Sect. 2 is:

$$PersonalName(\text{“}Androutsos\text{”}) \sqcap (DateofCreation \sqcap \text{TEST-C}(gt_{\text{“}1887\text{”}}))$$

2. The obtained DL expressions are *unfolded* by iteratively substituting the involved AP-derived concepts (**Tbox**-view part) with their constituent primitive ones (**Tbox**-schema part). For instance, **Q1** is rewritten as follows:

$$(Actor(\text{“}Androutsos\text{”}) \sqcup Owner(\text{“}Androutsos\text{”})) \sqcap (\exists happenedIn^{-1}.\exists ofKind.Kind(\text{“}Creation\text{”}) \sqcap \text{TEST-C}(gt_{\text{“}1887\text{”}}))$$

3. The *final expression* for Z39.50 query answers (C_{Answer}) is then obtained by introducing the corresponding path expressions (P_{AP}) for the involved primitive concepts and by considering in the answer only the individuals of the central concept (C_C):

$$C_{Answer} = \{a \in \mathcal{O}_\Sigma \mid \Sigma \models (MuseumObject(a) \sqcap ((\exists hasEvent.(\exists hasActor.Actor(\text{“}Androutsos\text{”})) \sqcup \exists ownedBy.Owner(\text{“}Androutsos\text{”})) \sqcap \exists hasEvent.(\exists happenedIn^{-1}.\exists ofKind.Kind(\text{“}Creation\text{”}) \sqcap \text{TEST-C}(gt_{\text{“}1887\text{”}}))\}$$

The above translations are considered under a *canonical form* of a conjunction of concept expressions such as $C(a)$ (primitive concept), $\neg C(a)$, $\exists R_1 \dots R_n.C(a)$, $\forall R_1 \dots R_n.C(a)$.

It is worth noticing that for Z39.50 queries using full text APs like *Any*, our rewriting will result in a DL expression comprising the paths to the central concept of all the source schema concepts. This rewriting essentially captures the translation at the source schema level [22] of the generalized path expressions (e.g., $MuseumObject(y)$ $@P(x)$) involved in the AP *Any*. Indeed, the resulting expression corresponds to a set of standard queries, i.e., without path variables. Furthermore, before evaluating it with source data, this expression is a subject of optimization by the wrappers taking into account concept subsumption. This is quite useful, especially for composite APs whose unfolding results in expressions comprising several times the same primitive concepts (e.g., *TemporalPoint* will be included in the translation of all temporal APs), and will be addressed in the next section.

5 Advanced Z39.50 wrapping services

In Sect. 3 we showed the benefits of modeling Z39.50 AP mappings as DL concepts (i.e., views) in order to formally validate their consistency. In this section, we focus on the capability of DL-based wrappers to reason about the relationships between the AP views as well as between these views and Z39.50 queries (also represented as DL concepts). Specifically, we show: (i) how a flat Z39.50 list of APs can be organized in a subsumption taxonomy thus rendering their source-specific conceptual structure; and (ii) how Z39.50 queries can be optimized with respect

to their intentional semantics without accessing actual source data (virtual **Abox**).

5.1 Conceptual structuring of flat Z39.50 vocabularies

Despite the simplified world view of information as a flat list of APs, Z39.50 profiles are usually developed according to an implicit conceptual structure of the information requested by the users. Indeed, the APs defined in a profile represent real world entities for a particular application, function, or community, at various abstraction levels and with different relationships between them. For example, in the CIMI-AQUARELLE profile [23, 53] we can observe a wide range of APs: from very abstract APs like *Any*, to general ones like *What*, *Who*, *When*, and *Where*, (the 4W APs) until more specific like *Date* or *DateOfCreation*. Making explicit their relationships in the context of a specific source, is very useful for both end-users and third-party metadata providers. It essentially allows one to understand why the conceptual structures of information in a source and a profile differ in order to improve the design of APs, query precision, interpretation of results, etc. This is one of the main motivations of ongoing projects (e.g., the Mozilla RDF/Z39.50 Project[10]) aiming at the integration of Z39.50 with advanced Web metadata standards (i.e., supporting schemas with classes/properties hierarchies) like RDF [13, 36].

We rely on the DL *Subsumption Checking* reasoning service to organize in a taxonomy the derived concepts capturing the AP mappings for a source. For instance, given the definition of *Date* and *DateOfCreation* (see Sect. 3) it can be inferred that $DateOfCreation \dot{\leq} Date$. In the simplest case, the subsumption relationships are hand-crafted by wrappers administrators in the definitions of composite AP concepts, e.g., the 4W APs.

Figure 3 illustrates the subsumption taxonomy of several CIMI-AQUARELLE APs as they are mapped to our example source (**Tbox**-*view part*). This taxonomy serves as advanced knowledge support about wrapped sources (i.e., metadata) which can be exploited online or offline. In both cases the Z39.50 `Explain` service[11] can be used in order to retrieve the mapping of a specific AP or of the whole **Tbox**-*view part*. Note that exchanging source metadata is not a simple task due to the different technologies (DBMS, KBS, etc.) employed by the sources and the various implementation choices made by wrapper administrators. We believe that a DL concept language can also be used to facilitate metadata retrieval (i.e., AP mappings) in a way commonly understood by all clients and independent from the underlying source/wrapper technology.

5.2 Intelligent query processing

In Sect. 4 we have seen that DL concept languages used to capture the schema of a source and define Z39.50 APs mappings as views on top of it, can also be employed to express the Z39.50 queries against these views. Not surprisingly Z39.50 queries can then be classified into the concept taxonomy using the subsumption relationships between them and the other primitive or derived concepts (**Tbox**-*query part*). The first benefit from this classification is to determine if a Z39.50 query can be effectively evaluated against the existing source schema and AP views. Indeed, after the translation of Z39.50 queries into a canonical DL form, wrappers are able to check whether the description (intention) of a query is contradictory without accessing the source data (virtual **ABox**). For instance, the following query can be detected as inconsistent since it uses the AP *ProtectionStatus* mapped to the *bottom* concept.

Q4: `PersonalName` = "Androutsos" `and` `ProtectionStatus` = "Preserved"

If now a query is semantically well-defined it can be appropriately classified by determining the set of its immediate *subsumers* and *subsumees*, i.e., the concepts found above or below in the taxonomy. Note that we are limited here to queries without comparison predicates (e.g., $<$, $>$, etc.). This classification opens interesting optimization opportunities since it induces a set of semantic transformations in order to locate the exact place of concepts in the taxonomy [6]. Consider, for instance, the following query where the derived concept *Who* subsumes *PersonalName* (see Fig. 3):

Q5: `PersonalName`="Androutsos" `or` `Who`="Androutsos"

Q5 will be rewritten into the following semantically equivalent query that will be actually executed by the source:

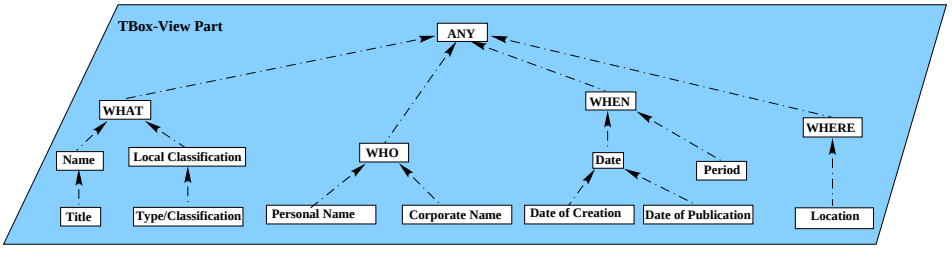

Fig. 3. Structuring a flat vocabulary of Z39.50 APs

[10] www.mozilla.org/rdf/doc/z3950.html

[11] A service allowing Z39.50 clients to retrieve metadata about servers

Q5': `Who` = "Androutsos"

Recall that the result of Z39.50 queries contains only individuals from a central concept (C_C) like *MuseumObject*. Therefore Z39.50 queries like **Q5** will be always classified under C_C defined in the **Tbox**-*view part*. This enables an intelligent caching of query results [3, 29, 39] by the wrappers and a consequent optimization of Z39.50 queries. If the concept representing a query is found to be equivalent to one already existing in the taxonomy, the interpretation of that concept can be returned as an answer set instead of evaluating it. This is the case of **Q5** assuming that the equivalent query **Q5'** has been previously evaluated and cached. Alternatively, the interpretations of all the immediate subsumers have to be checked against the query conditions. This is extremely useful, since Z39.50 is a stateful protocol and the results of previously executed queries can be kept in the server in order to be reused by subsequent queries. Consider the following example:

Q6: `Q5' and When` = 1815

In this case **Q5'** subsumes **Q6** and only the second part of the query needs to be executed by the source (intersection is performed locally by the wrapper). Finally, the results of **Q6** could also be cached in the wrapper. This implies that the cached interpretation of concept **Q5'** will now contain only its proper individuals, i.e., those not belonging to the interpretations of its immediate subsumees like **Q6**. Note that supporting several query answer sets proves to be quite expensive with current implementations of Z39.50 wrappers [10, 51, 52] since they rely on results replication.

The Z39.50 query processing and optimization techniques presented in this paper raise simplified versions of well-studied problems in the databases and knowledge representation literature. More precisely, the DL query subsumption checking corresponds to the well-known problem of *query containment* [20] while the optimization we propose can be considered as a form of *query answering* using views [41] where cached queries play the role of materialized views. Since we are limited to queries using standard DL unary and binary relations (i.e., without Horn rules and built-in predicates) both problems are decidable [15, 16], while the complexity of subsumption and instance checking of our language is PSPACE [8, 48]. Finally, as proven in [25] more interesting complexity results may be achieved for both reasoning services under the *closed domain assumption*.

6 The DL-based Z39.50 wrapper toolkit

The architecture of the DL-based toolkit we have developed [54] is shown in Fig. 4. It is composed of the following five modules:

Module 1 is responsible for *network communication* with the client and is based on the *Yaz* toolkit [33].

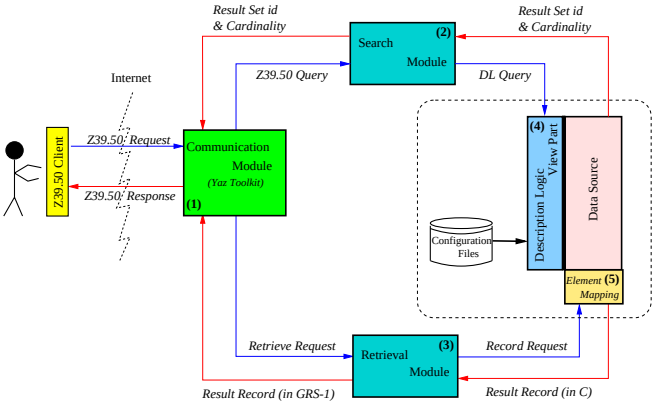

Fig. 4. The Z39.50 wrapper toolkit architecture

When it receives a search request it decodes it into appropriate C structures. More specifically, it produces the *syntax tree* of the query that is included in the search request and sends it to **Module 2**. When a response has to be sent back to the client, this module is responsible for the transformation of the answer to the appropriate network format.

Module 2 is used only during the *search* process. When it receives the syntax tree of a Z39.50 query, it translates it to a preliminary DL expression (see Sect. 4) that is sent to **Module 4**. After the query execution, it receives the *id* and the cardinality of the result set (not the data themselves) and forwards this information to **Module 1** to be sent back to the client.

Module 3 is used only during the *retrieval* process. After receiving a Z39.50 result set *id* it communicates with **Module 5** to get the retrieved records in the form of C++ structures. The task of **Module 3** is then to encode the returned C++ structures in one of the record formats defined in the Z39.50 profile (i.e., GRS-1, USMARC or XML) in order to send the retrieved records back to **Module 1**.

Modules 4 and **5** essentially form the *DL-based wrapper* for the underlying source (see dotted line in Fig. 4). **Module 4** loads the source schema and the AP mappings (**Tbox**) from a configuration file while the data reside in the source (virtual **Abox**) and can only be cached in the DL system. When it receives a DL query from **Module 2**, it rewrites it according to the defined AP mappings and the paths to central concept of interest (see Sect. 4) and forwards the resulting expression for evaluation to the underlying source. Finally, **Module 5** converts the retrieved objects of the central concept by taking into account the mappings of the Z39.50 Record Elements to the source data. Although not presented in this paper, these mappings are defined similarly to the APs.

All modules are operational while **Module 4** actually supports the DL-based Z39.50 query rewriting and sources built on top of the SIS-Telos [24]. For the pur-

poses of AQUARELLE and CIMIZit projects, we have developed from scratch only the DL *Instance Checking* service (see [54] for more details). However, we are currently working on the redesign of **Module 4** using available DL systems that support the constructs of our mapping language (e.g., CICLOP, CRACK). Due to the similarities between the DL and SIS-Telos query models, the translation of the resulting DL query expressions into our cultural source is straightforward. We plan to extend this interface of **Module 4** for other data source technologies, especially relational and object DBMSs (SQL, OQL), as already studied in [9, 12, 35].

To conclude, the modular architecture of the proposed toolkit allows one to significantly reduce wrapper development and maintenance costs. First, the DL-based **Module 4** can be reused in order to wrap the same source according to multiple, possibly overlapping profiles (e.g., AQUARELLE-CIMI and Dublin Core). This obviates the need to merge different Z39.50 profiles into one, in order to be supported by the existing wrappers. In our approach, the profile becomes a characteristic of the client query, rather than a characteristic of the source. Second, the same Z39.50 server can support several wrapped sources. This is due to the fact that **Modules 1**, **2**, and **3** need not be aware of the Z39.50 APs (or Element) mappings. This information is requested only by **Module 4**, i.e., the source wrapper. Hence, a server can support simultaneously sources of different technology, as well as Z39.50 profiles with different APs mappings in each data source.

7 Related work

Wrapping information sources is an essential step in order to tackle data heterogeneity issues (e.g., schematic, syntactic, system, etc.) in different application contexts: digital libraries, Web sites and portals, information integration systems, etc. The various wrapping approaches differ on: a) the *nature of data* provided by the underlying sources (e.g., from well-structured data in databases, loosely-structured data in document bases until unstructured data on the Web); b) the *middleware model* on which queries as formulated and result data are returned (e.g., structured or semi-structured models); c) the *supported functionality* (e.g., mappings, query translation, caching, etc.). In most of the cases, wrappers are used to transform source data from their native format and structure into a more expressive middleware model for further processing (e.g., analysis, consolidation, etc.). In our context, we are obliged to do the opposite, i.e., transform semantically and structurally rich data from DBMS or KBS sources into a flat Z39.50 information model. None of the existing approaches has addressed this issue.

In digital libraries, wrappers are used to adapt an existing search facility of a source to a specific retrieval protocol like Z39.50. Several prototype and commercial Z39.50 servers/wrappers are available[12] for sources built on top of an IRS, e.g., Bull Mistral [52], SSL Index+ [52], Information Dimensions Basis Plus [27], etc. Since there is no significant mismatch between the Z39.50 and the underlying IRS models (i.e., both support Boolean queries on flat textual records), query translation is straightforward. It essentially requires mapping the Z39.50 APs to the source record fields (i.e., queried attributes) and retrieval facilities (i.e., right/left truncations, proximity operations, etc.) using a configuration file. When such a mapping is not possible, the corresponding AP is not supported by the Z39.50 server. Compared to our approach, the supplied mapping languages are quite poor, since each AP can be mapped only to one queried field of the source (i.e., no fields combinations are possible) while unsupported APs lead to query failures.

More interesting mapping languages are provided by Z39.50 servers/wrappers for ODBC-compliant RDBMS sources [51]. They essentially use the SQL view mechanism to define the AP mappings as views over the source data. In this way, an AP can be mapped to several database fields combined by appropriate joins. Information about each AP view is stored in special-purpose tables. Dedicated tables are also used in order to store the query results for subsequent use, given the stateless nature of SQL. However, as in the previous cases, unsupported APs lead to embarrassing query failures. On the other hand, our DL-based wrapping allows one not only to capture a wide range of Z39.50 mapping cases (including unsupported APs) but also to reason (e.g., for consistency, subsumption) about the defined AP views as well as the issued Z39.50 queries on top.

It should be stressed that this kind of added-value services is usually provided by intelligent information integration systems [2, 5, 17, 28, 30, 42]. In these systems, domain modeling and reasoning issues are addressed at a higher layer in the integration architecture, i.e., by *mediators*, using an appropriate KRRS middleware technology (e.g., DL). Compared to this work, our context is quite different: (i) Z39.50 wrapping involves only one source at a time (vs. mediation of several sources); and (ii) Z39.50 world view of information is intrinsically flat (vs. middleware structured models). As a matter of fact, the main contribution of this paper was to push some of the intelligent mediator's functionality to the Z39.50 wrappers in a standard client-server architecture.

The whole Dl-based framework for Z39.50 wrapping is reminiscent of the *Global as View* (GAV) approach proposed in information integration systems [40]. The definition in our approach of the AP mappings over source concepts, is similar to the specification for each relation R in the mediated (global) schema of a query over the source relations, indicating how to obtain R's tuples from the sources. The main advantage of the GAV approach is that

[12] http://lcweb.loc.gov/z3950/agency/register/entries.html

query reformulation is very simple, because it reduces to rule unfolding. However, adding new sources to the data integration system is non-trivial. In particular, we need to consider the possible interaction of the new source with each of the existing sources, and this limits the ability of the GAV approach to scale to a large collection of sources. In contrast, in our Z39.50 wrapping context, the global concepts, i.e., the APs, have been defined in advance, independently from the underlying source schemas. Therefore, the established mappings do not enrich the global schema with new concepts while they are used exclusively by the wrappers and not the mediator (avoiding the scalability problems of the GAV approach).

8 Conclusion and future work

In this work we have addressed the declarative specification of Z39.50 wrappers. We have presented a wrapper generation toolkit based on DL concept languages to map the Z39.50 world view of information to the underlying source data structure and semantics. The proposed DL mapping language offers a number of advantages: (i) the required views over source data can be easily defined while a wide range of Z39.50 translation cases can be expressed; (ii) it comes equipped with formally verifiable properties allowing to check the consistency of the defined views and therefore ensure the quality of the retrieved data; (iii) it enables reasoning about the relationships between these views and thus rending explicit to Z39.50 profile developers, end-users, etc., the conceptual structure of the Z39.50 vocabularies for a specific source; and (iv) it can serve to translate Z39.50 queries, which opens interesting opportunities for semantic query optimization and caching of results, exploiting as much as possible the stateful nature of the protocol.

Currently, the developed toolkit supports only the DL *Instance Checking* service for evaluating queries against sources built on top of SIS-Telos [24]. We are planning to provide full-fledged DL reasoning services by integrating our toolkit with one of the available DL systems[13] (studying expressiveness of offered languages, existing APIs support etc.). Furthermore, we intend to validate our approach with several Z39.50 profiles and extend the wrapping facilities to other data source technologies (e.g., DBMS, IRS, etc.). Last but not least, we plan to apply the ideas presented in this paper at a higher level of information integration, in order to build intelligent Z39.50 mediators instead of wrappers. More precisely, we are currently working on the problem of mapping detection by developing appropriate algorithms for assisting the user in the mapping definition task, offering automation and validation services.

Acknowledgements. We are grateful to the AQUARELLE and CIMI Consortium for their technical support. We also thank A. Analyti, D. Plexousakis, Y. Tzitzikas, and M. Döerr for helpful comments on preliminary versions of this paper.

[13] www.ida.liu.se/labs/iislab/people/patla/DL/systems.html

References

1. ANSI/NISO: Z39.50 (versions 2 and 3) Information Retrieval: Application Service Definition and Protocol Specification, 1995
2. Arens, Y., Chee, C.Y., Hsu, C.-N., Knoblock, C.A.: Retrieving and Integrating Data from Multiple Information Sources. International Journal of Cooperative Information Systems 2(2):127–158, 1993
3. Ashish, N., Knoblock, C.A., Shahabi, C.: Intelligent Caching for Information Mediators: A KR Based Approach. In: Proc. of the 5th Workshop. KRDB'98, Seattle, Washington, 1998, pp. 3.1–3.7
4. Baader, F., Bürckert, H., Heinsohn, J., Hollunder, B., Müller, J., Nebel, B., Nutt, W., Profitlich, H.: Terminological Knowledge Representation: A Proposal for a Terminological Logic. In: Nebel, B., Luck, K., Peltason, C. (eds.): Proc. of the First Int. Workshop on Terminological Logics. Dagstuhl, Germany, 1991. DFKI
5. Bergamaschi, S., Castano, S., Vincini, M.: Semantic Integration of Semistructured and Structured Data Sources. SIGMOD Record Special Issue on Semantic Interoperability in Global Information 28(1): March 1999
6. Bergamaschi, S., Sartori, C., Vincini, M.: DL Techniques for Intensional Query Answering in OODBs. In: Proc. of the 2nd Workshop. KRDB'95. Bielefeld, Germany, September 1995
7. Borgida, A.: Description Logics for Querying Databases. In: Proc. of the 1st Int. Workshop on Description Logics. Bonn, Germany, May 1994, pp. 95–96
8. Borgida, A.: Description Logics in Data Management. IEEE Transactions on Knowledge and Data Engineering 7(5):671–682, October 1995
9. Borgida, A., Brachman, R.J.: Loading Data into Description Reasoners. In: Proc. of the ACM SIGMOD Conf. on Management of Data. SIGMOD'93. June 1993, pp. 217–226
10. Bouthors, V., Dupuis, J., Huu, N.T.: Z39.50 Gateway for Mistral and ARF DTD Specification. Aquarelle project, deliverable 5.2, INRIA, France, September 1997
11. Brachman, R., McGuinness, D., Schneider, P.P., Resnick, L.A., Borgida, A.: Living with CLASSIC: When and How to use a KL-ONE-like language. In: Sowa, J.F. (ed.): Principles of Semantic Networks – Explorations in the Representation of Knowledge, Morgan Kaufmann, 1991, pp. 401–456
12. Bresciani, P.: Querying Databases from Description Logics. In: Proc. of the 2nd Workshop. KRDB'95. Bielefeld, Germany, September 1995
13. Brickley, D., Guha, R.V.: Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation. Technical Report CR-rdf-schema-20000327, W3C, Available at http://www.w3.org/TR/rdf-schema, March 27, 2000
14. Buchheit, M., Donini, F., Nutt, W., Schaerf, A.: Terminological Systems Revisited: Terminology=Schema + Views. In: Proc. of the First Workshop. KRDB'94. Saarbrücken, Germany, September 1994
15. Calvanese, D., Giacomo, G. De, Lenzerini, M.: On the Decidability of Query Containment under Constraints. In: Proc. of the 17th ACM Symposium on Principles of Database Systems. PODS'98, June 1998, pp. 149–158
16. Calvanese, D., Giacomo, G. De, Lenzerini, M.: Answering Queries Using Views in Description Logics. In: Proc. of the 6th Int. Description Logic Workshop. DL'99. Linköping, Sweden, July 1999
17. Calvanese, D., Giacomo, G. De, Lenzerini, M., Nardi, D., Rosati, R.: Description Logic Framework for Information Integration. In: Proc. of the 6st Int. Conference on Principles of Knowledge Representation and Reasoning. KR'98. Povo-Trento, Italy, June 1998, pp. 2–13
18. Calvanese, D., Giacomo, G. De, Rosati, R.: A Note on Encoding Inverse Roles and Functional Restrictions in ALC Know-

ledge Bases. In: Proc. of the 5th Int. Description Logic Workshop. DL'98. Povo-Trento, Italy, June 1998, pp. 11–20
19. Calvanese, D., Lenzerini, M., Nardi, D.: Description Logics for Conceptual Data Modeling. In: Chomicki, J., Saake, G. (eds.): Logics for Databases and Information Systems. Kluwer, 1998
20. Chekuri, C., Rajaraman, A.: Conjunctive Query Containment Revisited. In: Afrati, F., Kolaitis, P. (eds.): Proc. of the Sixth Int. Conference on Database Theory. ICDT'97. Delphi, Greece, January 1997, pp. 56–70
21. Christophides, V., Abiteboul, S., Cluet, S., Scholl, M.: From Structured Documents to Novel Query Facilities. In: Proc. of the ACM SIGMOD Conf. on Management of Data. SIGMOD'94. Minneapolis, Minnesota, May 1994, pp. 313–324
22. Christophides, V., Cluet, S., Moerkotte, G.: Evaluating Queries with Generalized Path Expressions. In: Proc. of the ACM SIGMOD Conf. on Management of Data. SIGMOD'96. Montreal, Canada, June 1996, pp. 413–422
23. CIMI, The CIMI Profile Release 1.0h: A Z39.50 Profile for Cultural Heritage Information. Technical report, Consortium for the Computer Interchange of Museum Information, Available at http://www.cimi.org/documents/HarmonizedProfile/HarmonProfile1.htm, November 1998
24. Constantopoulos, P., Doerr, M.: The SIS System: A brief presentation. ICS-FORTH, http//www.csi.forth.gr/isst, May 1993
25. Donini, F.D., Lenzerini, M., Nardi, D., Schaerf, A.: Reasonig in Description Logics. In: Brewka, G. (ed.): Proc. of the Fourth Int. Conference on Principles of Knowledge Representation and Reasoning. KR'96. Studies in Logic, Language and Information, CLSI Publications, 1996, pp. 193–238
26. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W.: Queries, Rules and Definitions as Epistemic Sentences in Concept Languages. In: Lakemeyer, Gerhard, Nebel, Bernhard (eds.): Foundations of Knowledge Representation and Reasoning. Vol. 810 of Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin, 1994, pp. 113–132
27. Finsiel, Zeta suite. Available at http://zeta.tlcpi.finsiel.it/zetasuite/, Italy, 1998
28. Goasdoué, F., Lattes, V., Rousset, M.-C.: The Use of CARIN Language and Algorithms for Information Integration: The PICSEL Project. In: Proceedings of the Second International and Interdisciplinary Workshop on Intelligent Information Integration, 1998
29. Goni, A., Illarramendi, A., Mena, E., Blanco, J.M.: An Optimal Cache for a Federated Database System. Journal of Intelligent Information Systems (JIIS) 9(2):125–155, 1997
30. Goni, A., Mena, E., Illarramendi, A.: Information Modelling and Knowledge Bases, chapter Querying Heterogeneous and Distributed Data Repositories using Ontologies, IOS Press, 1998, pp. 19–34
31. Horrocks, I., Sattler, U.: A Description Logic with Transitive and Inverse Roles and Role Hierarchies. In: Proc. of the 5th Int. Workshop on Description Logics. DL-98. Povo-Trento, Italy, June 1998
32. Hustadt, U.: Do we need the Closed World Assumption in Knowledge Representation? In: Proc. of the 1st Workshop KRDB'94, Saarbrücken, Germany, September 1994
33. Index Data, Available at http://www.indexdata.dk/yaz. *Yaz User's Guide and Reference Manual*, version 1.4 edition, 1997
34. Janney, K., Sledge, J.: A User Model for CIMI Z39.50 Application Profile . Available at http://www.cimi.org/documents/Z3950_app_profile_0995.html, September 1995
35. Kessel, T., Schlick, M., Speiser, H.-M., Brinkschulte, U., Vogelsang, H.: C3L+++: Implementing a Description Logics System on Top of an Object-Oriented Database System. In: Proc. of the 3nd Workshop. KRDB'96. Budapest, Hungary, August 1996
36. Lassila, O., Swick, R.: Resource Description Framework (RDF) Model and Syntax Specification, W3C Proposed Recommendation. Technical Report REC-rdf-syntax-19990202, W3C, Available at http://www.w3.org/TR/REC-rdf-syntax, February 22, 1999
37. LC, Z39.50 Profile for Access to Digital Collections. Technical report, Library of Congress, Available at http://lcweb.loc.gov/z3950/agency/profiles/collections.html, 1996
38. LC, Application Profile for the Government Information Locator Service (GILS). Technical report, Library of Congress, Available at http://www.gils.net/prof_v2.html, 1997
39. Lee, D., Chu, W.W.: Semantic Caching via Query Matching for Web Sources. In: Proc. of the Eighth Int. Conf. on Information and Knowledge Management. CIKM'99. Kansas City, Missouri, November 1999
40. Levy, A.: Combining Artificial Intelligence and Databases for Data Integration. To appear in a special issue of LNAI: Artificial Intelligence Today; RecentTrends and Developments, 1999
41. Levy, A.: Answering Queries Using Views: a Survey, submitted for publication 1999
42. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. In: Proc. of the Int. Conf. on Very Large Databases. VLDB'96. Bombay, India, September 1996, pp. 251–262
43. Levy, A.Y., Rousset, M.C.: Using Description Logics to Model and Reason About Views. In: Wahlster, W. (ed.): Proc. of the 12th European Conf. in Artificial Intelligence. ECAI'96. Budapest Hungary, John Wiley & Sons, Ltd., August 1996
44. Michard, A., Christophides, V., Scholl, M., Stapleton, M., Sutcliffe, D., Vercoustre, A-M.: The Aquarelle Resource Discovery System. Journal of Computer Networks and ISDN Systems 30(13):1185–1200, August 1998
45. Moen, W.E.: Accessing Distributed Cultural Heritage Information. Comm. of ACM 41(4):45–48, April 1998
46. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: Representing Knowledge About Information Systems. ACM Transactions on Information Systems 8(4):325–362, 1990
47. Nebel, B.: Terminological Cycles: Semantics and Computational Properties. In: Sowa, J.F. (ed.): Principles of Semantic Networks. Morgan Kaufmann, San Mateo, 1991, pp. 331–362
48. Schaerf, A.: Query Answering in Concept-Based Knowledge Representation Systems: Algorithms, Complexity, and Semantic Issues. Ph.D. thesis, Dipartimento di Informatica e Sistemistica Univerita di Roma:La Sapienza, 1994
49. Schild, K.: The use of Description Logics as Database Query Languages. In: Proc. of the 2nd Workshop KRDB'95, Bielefeld, Germany, September 1995
50. Sheth, A.: Changing Focus on Interoperability in Information Systems: From System, Syntax, Structure to Semantics. In: Goodchild, M.F., Egenhofer, M.J., Fegeas, R., Kottman, C.A. (eds.): Interoperating Geographic Information Systems. Kluwer Academic Publishers, February 1999
51. Signore, O., Loffredo, M.: Z39.50-SQL Gateways: Technical Description. Aquarelle project, deliverable 5.1, CNR-CNUCE, Italy, April 1997
52. SSL, Z39.50 version of Index+: Technical Description. Aquarelle project, deliverable 5.3, System Simulation Ltd, UK, October 1997
53. SSL, Aquarelle Z39.50 Profile. Technical report, Aquarelle: The Information Network on Cultural Heritage, Available at http://aqua.inria.fr/Aquarelle/Public/EN/profile-2.0.html, May 1998
54. Velegrakis, Y.: Declarative Specification of Z39.50 Wrappers using Description Logics. Technical Report FORTH-ICS-TR-225, Computer Science Institute, Foundation of Research and Technology (ICS-FORTH) – Hellas, July 1998. M. Sc. thesis, Computer Science Department, University of Crete